# Approaches to Improve Reliability of Service Composition

Jörg Hohwiller, Diethelm Schlegel, and Gregor Engels

Capgemini, CSD Research, Berliner Str. 76, 63065 Offenbach, Germany
{joerg.hohwiller,diethelm.schlegel}@capgemini.com, engels@upb.de

**Abstract.** Nowadays, enterprises realize functionality as systems that are composed of services. This includes even mission critical parts of their business. Hence, the reliability of such systems including their composition and services is increasingly important. However, it is a challenge to establish a high reliability in this context because distribution of functionality increases the potential points of failure. Different approaches exist to increase reliability but they typically act on a restricted scope like network layer or software design. In order to obtain better results, it is often necessary to combine multiple approaches depending on the actual situation and the requirements. This paper classifies commonly used approaches according their scope and rates their effects on the reliability. Thereby, it supports the selection of approaches to improve reliability and finally helps to find a suitable solution for a given situation.

**Keywords:** Reliability, SOA, Composition, Service, Quality of Service.

## 1 Problem

*Services* are functionalities that are exposed over networks using standardized interfaces. They are the base of *service oriented architectures* (SOA). Enterprises implement even critical business logic by services. In the past, the majority of services was implemented internally. Nowadays, it can be observed that the use of external services is growing fast. On the other hand, *service composition* combines multiple existing services to a new one which offers a more complex functionality. As shown in figure 1, the realization uses a wide range of different hardware and software components including networks. [3] deals with the prediction of the resulting reliability of the composite service.

Independent of whether a service is composed or not, consumer expect its reliability in accordance with the concrete usage conditions. Generally, *reliability* is defined as the ability of a service to perform its required functions under stated conditions for a specified time interval [10] (e.g. an availability of 99.45% for a 7/24 service typically means it must be fully functional at least 363 days a year). It contains four aspects: maturity, fault tolerance, recoverability, and reliability compliance (see [7]).

*Service level agreements* (SLA) typically regard reliability with respect to *stated conditions*. To guarantee a SLA, systems must cope with this well defined

G. Pallis et al. (Eds.): ICSOC 2011, LNCS 7221, pp. 321–332, 2012.

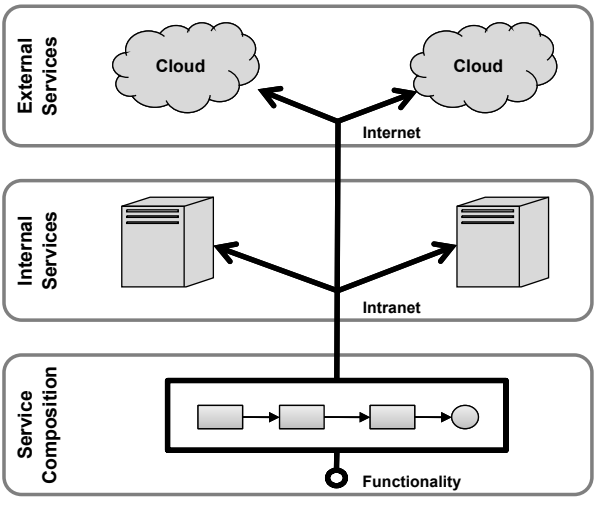


**Fig. 1.** Service Composition

set of *situations* that could be systematic or accidental, temporary or permanent, targeted or unintended. The importance of conditions depends on the concrete usage scenarios. Several users of the same service may even have different SLAs depending on their actual conditions and payments.

In many cases, the effects of service faults on the business are disastrous. So, measures must be taken for reducing the risk of service failures. There are several commonly used techniques like failover mechanisms or redundancy to enhance service reliability. But their value depends on the environment and situation. When designing a system, architects must be familiar with these interrelations to choose the right concepts. For example, it has no major effect to replicate databases when the most critical problems arise from overloaded networks.

In the following sections, situations with negative impact on the behaviour of composed services are classified. Some approaches for reliability improvement are described and rated concerning their effects in these situations. This will give architects an advice to decide which techniques he shall use for individual usage scenarios and possible fault situations. Hereby, our results mainly result on empirical experiences collected in customer projects.

## 2 Situations and Scopes

The overall reliability of a composed service is affected by a large variety of situations. This section introduces two dimensions that classify situations into scopes. Thus, it is possible to rate the capabilities of approaches to increase the service reliability in individual situations (see section 3).

The first dimension consists of different levels. of influence It ranges from the central service composition up to external services in the cloud. In the same way, the approaches to improve the reliability act on one or more of these levels. The

following levels are distinguished in this paper. They correspond with the three layers represented in figure 1.

- The *composition level* summarizes the logical design, the model, and the physical implementation of resulting composed services, e.g. by using *business process engines* (BPE). It also includes the interfaces that make the resulting services available.
- The *internal level* contains hardware, network components, operating systems (OS), runtime environments, and the implementation of the services used by the composition.
- The *external level* collects hardware and software components external to the company. The main difference to the first two levels consists in the fact that there are only few or no possibilities to influence these components.

As a second dimension, the situations are grouped by provoking causes leading to deteriorated reliability of service compositions. There are four condition clusters regarded in this paper.

- *Overloads* are characterized by the presence of too many or too complex requests. This implies that components are used more intensely than they are designed for. If the situation is provoked intentionally it is called a *denial of service attack*.
- *Failures* sum up all situations where hardware or software components do not perform the way they are designed to. Examples are unstable operating systems, software abortions, or complete breakdown of hardware (servers, network, power-supply, internet connection, etc.).
- As opposed to failures, *design mistakes* cover all situations where components sure perform well but their behaviour is not consistent with their requirements. They are more systematic and occur deterministically. Reliable service compositions are expected to handle such conditions properly.
- *Attacks* contain situations where components are out of order due to external or internal enemies aimed at bringing wohle systems or single components down.

Table 1 shows examples of situations in order to clarify the two dimensions described above. The same table layout will be used in section 3 to rate the approaches.

## 3 Solution

This section shows approaches aimed at improving the reliability of services and compositions. The solution is created by choosing the right combination of approaches for the given requirements. Each approach is summarized by a table showing the effects on reliability for the classes as in table 1 (see section 2). Each cell shows a rating of the impact for the corresponding situations. A gain is marked as +. On the other hand a - indicates a negative and an empty cell no relevant influence.

**Table 1.** Situations with Impact on Reliability

| Level \ Cond. | Overload | Failure | Design | Attack |
|---|---|---|---|---|
| Composition | ▪ Too many requests<br>▪ Too complex models | ▪ Model abortion<br>▪ Race condition<br>▪ Dead lock | ▪ Wrong system architecture<br>▪ Incorrect service composition | ▪ Distributed Denial of Service<br>▪ Unauthorized access |
| Internal | ▪ Too many invocations<br>▪ CPU load too high | ▪ Server crash<br>▪ Power outage<br>▪ Ressource leak<br>▪ Service failure | ▪ Wrong config<br>▪ Insufficient hardware<br>▪ Unsuitable network architecture | ▪ Trojan Horse<br>▪ Viruses |
| External | ▪ Not enough bandwidth<br>▪ Overload of external service | ▪ Internet (access) problems<br>▪ External service not available | ▪ Provider SLA not sufficient<br>▪ Ext. Service not suitable | ▪ Distributed Denial of Service<br>▪ Worms |

### 3.1 Load Balancing

An obvious approach to increase reliability is to have multiple redundant instances. This applies to all technological levels in order to avoid single points of failure.

- *Composition:* multiple instances (physical hardware or virtual instances)
- *Internal services:* multiple instances of internal services, network redundancy
- *External services:* second ISP, redundant cloud services

Having redundant instances available in cold-standby will only help to avoid maintenance downtimes. In undesired conditions (failure and overload) availability can be increased by dynamic delegation of requests across multiple available redundant instances. This is achieved by a proxy as front-end to the service. Such proxy serve two demands.

- *Failover:* If one of the redundant instances becomes unavailable requests are delegated to another instance.
- *Load balancing:* The proxy distributes the load of incoming requests among the available instances. There are various strategies like round-robin [2].

There are existing hardware and software solutions to realize *load balancers* (proxy servers capable of load balancing and failover). Especially, this applies for HTTP as a de facto standard for the transport of services. The same approach can be used on external level in the other direction by clustering internet connections of different service providers.

A redundant design of service implementations can cause problems if the functionality is not read only. Write operations allowing modifications of underlying persistent data require synchronization of changes between redundant instances.

This is solved by concepts like master/slave replication or clustering of databases (e.g. see [1]). On the other hand the service logic should be designed stateless for good scalability.

The rating of this approach is summarized in table 2. It improves the reliability only if applied on the composition and on internal services because it is impossible to influence the redundancy the whole external equipment involved. Furthermore, design mistakes and attacks cannot be relieved by duplicating instances.

**Table 2.** Approach: Load Balancing

| | | | | |
|---|---|---|---|---|
| **Summary:** | Dynamically route requests to redundant instances according to load and presence | | | |
| **Preconditions:** | Stateless service or synchronization required | | | |
| **Advantages:** | - Simple and generic approach<br>- Established, evident gain of availability | | | |
| **Drawbacks:** | - High cost (e.g. duplicate hardware costs, increasing maintenance)<br>- Potential synchronization issues on *service level* | | | |
| **Level \ Cond.** | **Overload** | **Failure** | **Design** | **Attack** |
| **Composition** | + | + | | |
| **Internal** | + | + | | |
| **External** | | | | |

### 3.2 Quality of Service

As described in section 1 the requirements for reliability differ per usage scenario. Here we extend the approach *redundancy* (see 3.1). Instead of giving all requests the same priority in the load balancing, the quality of service (QoS) level depends per usage scenario. This mechanism can be applied on composition side when exposing the service interface and for the internal and external services. QoS can be achieved by different approaches.

- *QoS per Endpoint (Virtualized Services)* defines different endpoints (URLs) for each usage scenario that represent the same service logic but offer different QoS. Realizing this with entirely redundant hardware would be a waste of resources. Instead, the endpoints represent *virtualized services* that delegate to the same back-end (the redundant instances described in section 3.1). Each virtualized service is realized as separate load balancer (see 3.1) and can ensure a different QoS level. This also includes limiting the QoS by reducing the number of parallel requests or not involving all available redundant instances from the backend (see Figure 2). Different usage scenarios then use different virtual services. On *infrastructure level* it can be ensured that only the dedicated virtual service is reachable for a particular client. In advance to load balancing and failover (see 3.1) this approach is more robust. Overload conditions and attacks will only block a single usage scenario.

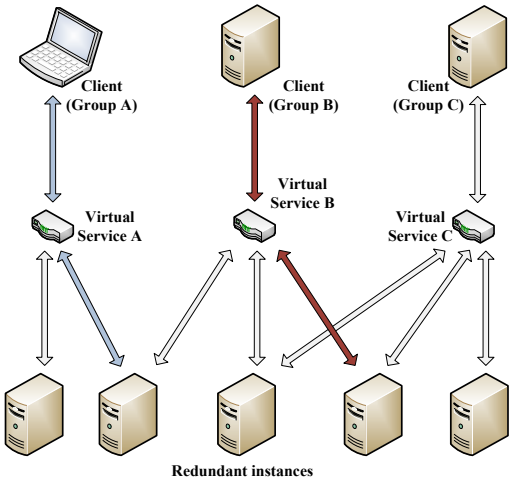


**Fig. 2.** QoS per Endpoint

- Typically, the user is related to the usage scenario that is associated to a particular QoS level (*QoS per User*). The user invoking the service needs to be identified and its according QoS level determined. If the service requires authentication the user identification arises implicit. Otherwise information as the IP-address have to be used. As such information can be manipulated additional security mechanisms may be required. If the appropriate QoS level is determined, the application logic (service facade) can route the request accordingly to one of the redundant back-end instances (e.g. analogue to *QoS per endpoint*). This should typically be addressed by an enterprise service bus (ESB). However, it is important to assure the chosen product is robust enough not to become a bottleneck.
- A very flexible approach is to send the QoS requirements with each service request (*QoS per Request*, see also [13]). The question is how much trust shall be given to the requester. In a closed and isolated world this approach will work well, but in an open and uncontrolled world (e.g. the public cloud) some control and restriction mechanisms have to be added on the provider side. Otherwise a DoS-attacker will have it even easier to produce high load. The main scenario of this approach is therefore a service that is used by a smaller set of clients over a secured channel. If requests are charged according to the required QoS a balance will establish. Such offering will typically require service level agreements and measurement (see [5] and [11]) for proper acceptance.

Table 3 shows the rating of this approach. Compared with failover mechanisms it copes with denial of service attacks and overload situations because only single QoS groups are affected. This even is relevant for external services. As a drawback, establishing appropriate QoS based redundancy is more expensive than simple failover mechanisms.

**Table 3.** Approach: Quality of Service

| **Summary:** | Dynamically route requests to redundant instances with different QoS-levels | | | |
|---|---|---|---|---|
| **Preconditions:** | Redundancy (load balancing and failover) | | | |
| **Advantages:** | - Availability even in case of overload / attack | | | |
| **Drawbacks:** | - High cost (e.g. multiple times of hardware costs, more maintenance)<br>- Potential synchronization issues on service level | | | |
| **Level** \ **Cond.** | **Overload** | **Failure** | **Design** | **Attack** |
| **Composition** | + | + | | + |
| **Internal** | + | + | | + |
| **External** | + | | | + |

### 3.3 Parallel Service Invocation

If the reliable service needs some functionality provided by another service and there is more than one possible service provider offering the needed function, the requesting service can decide what service it calls.

The service calls at least two of the possible providers, waits until the first result is received and aborts the other calls. If data is modified it is important that changes are performed only once. This implies the need for compensation, transactional behaviour or two phases commit (see [12]). The call is successful if at least one service provider is performing well. The time needed for the call is the minimum time of all providers (see [9] and [8]).

The more independent the service implementations are in respect of infrastructure and implementation, the higher will be the benefit of this approach. A disadvantage is the higher resource utilization. The network load increases by sending multiple requests in parallel. If adopted on the large scale service providers must cope with more requests in a given time interval. The approach is not useful against attacks and concerning the composed service itself.

### 3.4 Dynamic Service Composition

The main disadvantage of the redundant service invocation approach consists in the load it generates and the need for compensation or two phases commits. It is desirable to call only one of the service providers to save bandwidth. As a solution, *static service composition* is established by manually configuring the provider to use. This approach resembles the load balancing approach (see 3.1) with another service provider available as a backup.

As a drawback, failure and overload cannot be detected without delay and intervention. The solution is to choose the provider dynamically. The challenge is to find the service provider with the best reliability based on an algorithm. It must react on changing conditions fast and in an appropriate way. The general problem to find the best provider at any time cannot be solved, because it is

**Table 4.** Approach: Parallel Service Invocation

| | | | | |
|---|---|---|---|---|
| **Summary:** | Call more than one service provider in parallel | | | |
| **Preconditions:** | Critical service, ≥2 providers available, compensation or two phase commit, stateless service | | | |
| **Advantages:** | - Guarantees the best response behaviour at the given conditions | | | |
| **Drawbacks:** | - Generates overload on network, proxies and service implementation | | | |
| **Cond. / Level** | **Overload** | **Failure** | **Design** | **Attack** |
| **Composition** | | | | |
| **Internal** | - | + | | |
| **External** | - | + | + | |

impossible to predict the future conditions. Besides, regular checking of service availability would increase network load.

Optimizing dynamic service composition is known as NP-hard (see [14]). However, there are a lot of proposals for QoS aware service compositions based on integer programming, case-based reasoning, genetic algorithms (see [4]), and hybrid approaches (see [14]). Which of these will be appropriate depends on the given situation and structure. Sudden overload, attacks, or failure conditions will be recognized with delay and therefore negative impact on reliability. Additionally, it improves the behaviour in the context of design mistakes (composition and external services).

**Table 5.** Approach: Service Composition

| | | | | |
|---|---|---|---|---|
| **Summary:** | Call the service provider with best predicted QoS | | | |
| **Preconditions:** | Critical service, ≥ 2 providers available, probability for sudden overload or failure not too high | | | |
| **Advantages:** | - Improves the response behaviour by making some assumptions of the service provider conditions | | | |
| **Drawbacks:** | - Reacts on sudden overload or failure with delay | | | |
| **Cond. / Level** | **Overload** | **Failure** | **Design** | **Attack** |
| **Composition** | | | + | |
| **Internal** | + | + | | + |
| **External** | + | + | + | + |

### 3.5 Service Interface Granularity

All preceding approaches take services as given and add the appropriate mechanisms. Another way to improve reliability consists in better system design. It is

the preferred way when implementing a new composition including the internal services, but it can also be accomplished when re-designing consisting applications or selecting external services. This paper considers the service granularity. In any case, extremes must be avoided. Monolithically designed services as well as thousands of small services will lead to poor performance and reliability.

As stated in [6] coarser grained interfaces reduce the number of calls and at the same time network utilization. So, the impact of overload will be minimized. On the other hand, the interfaces will become more complex leading to increasing requirements and execution times as well as to higher design mistake rates. This also decreases the reliability against attacks. The probability to find alternative providers for functionalities decreases.

Finer grained interfaces result in leaner implementations. The chance to find redundant providers becomes higher which indirectly leads to more robust behaviour in overload or failure conditions. Transactional reliability can be achieved more easily if operations are less complex. At the same time, the increasing number of service calls leads to higher network utilization. If external providers are called the individual round trip delays of multiple invocations sum up reducing the performance compared to singe service calls. In summary it can be stated that finer grained services have more demands concerning the underlying network infrastructure.

**Table 6.** Approach: Service Granularity

| | | | | |
|---|---|---|---|---|
| **Summary:** | Choose the appropriate interface complexity and service granularity | | | |
| **Preconditions:** | Service composition not yet fixed | | | |
| **Advantages:** | - More reliable implementation, better chance to find redundant service implementations | | | |
| **Drawbacks:** | - Possible increase of network and proxy utilization | | | |
| **Cond. / Level** | **Overload** | **Failure** | **Design** | **Attack** |
| **Composition** | + | | + | + |
| **Internal** | + | | + | |
| **External** | + | | + | + |

### 3.6 Solutions for Usage Scenarios

Figure 3 shows an overview of the approaches and the resulting positive and negative impact on reliability. This implies solutions for the following usage scenarios of a service

– *Internal:* As long as the system is composed of services in the local network redundancy (3.1, 3.2) will be the adequate approaches to ensure sufficient reliability. Additionally, if the infrastructure is properly sized, it is preferable to choose a finer interface granularity (3.5). If external services are involved

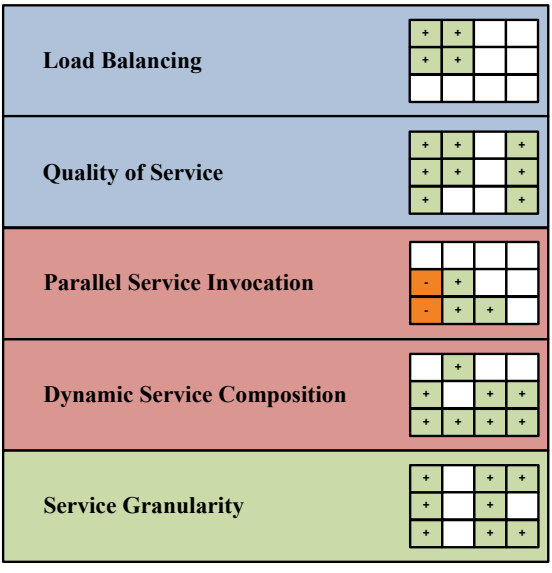


**Fig. 3.** Approaches to Improve Reliability

the approaches with respect to the parallel service invocation (3.3) or dynamic service composition (3.4) will become preferred solutions.

- *Business to Business:* If multiple partners must be provided by different QoS levels for the same service, one of the QoS based redundancy (3.2, 3.2) is the best approach. The interface granularity should be rather coarser grained (3.5) to reduce the network traffic.
- *Business to Customer:* If the service is also exposed to customers it is highly recommended to choose the QoS per Endpoint approach (3.2). Requests should be separated into QoS groups to be able to guarantee high service reliability for important users. Finer grained interfaces (3.5) would be slightly better in this case to prevent denial of service attacks by calling complex services.

Based on the results there are the following ruler of thumb. The usage scenarios and rules of thumb must be checked against the actual requirements before implementing them because their effects highly depend of the concrete scenario.

1. As long as services are composed internally redundancy should be established.
2. As long as cloud services are involved parallel invocation or dynamic service composition are appropriate.
3. Cloud services should tend to be coarser granular.
4. Internal services should be designed finer granular.

## 4 Conclusion and Further Work

This paper dealt with the question how to make a service composition more reliable, especially if external service providers are involved. Based on two dimensions several approaches were rated and summarized with respect to their impact on reliability. The concrete solution consists in selecting one or more approaches depending on the actual scenario.

Furthermore, the solution depends on the point of view. A service provider can improve the reliability by load balancing (3.1) and QoS mechanism (3.2) while a service requester will consider parallel invocation or dynamic service composition (3.4). This is summarized by exemplary usage scenarios and rules of thumb in section 3.6. After all, true service reliability can only be achieved if all technological levels are addressed.

Based on this paper possible future work includes analysing further approaches (e.g. architectural design questions) and extending dimensions (e.g. with conditions like duration and severity). Further, the methodology can be extended by incorporating security aspects or implementation costs. So, the ratings of the approaches will become even more useful.

## References

1. Database replication, `http://www.tech-faq.com/database-replication.html`
2. Bhavin Turakhia, P.K.: The compendium of load balancing strategies. Tech. rep., Wiki. Directi (2009)
3. Bocciarelli, P., DAmbrogio, A.: A model-driven method for describing and predicting the reliability of composite services. Software and Systems Modeling 10, 265–280 (2011), doi:10.1007/s10270-010-0150-3
4. Canfora, G., Di Penta, M., Esposito, R., Villani, M.L.: An approach for qos-aware service composition based on genetic algorithms. In: GECCO 2005: Proceedings of the 2005 Conference on Genetic and Evolutionary Computation, pp. 1069–1075. ACM, New York (2005)
5. Dobson, G.: Quality of service in service-oriented architectures. Tech. rep., Lancaster University (2004)
6. Friedl, B.: Zur optimalen Granularität von IT-Services - Eine Analyse relevanter ökonomischer Einflussfaktoren. In: Bernstein, A., Schwabe, G. (eds.) Proceedings of the 10th International Conference on Wirtschaftsinformatik, vol. 1, pp. 404–413 (2011)
7. International Organization for Standardization: ISO/IEC 9126. Software engineering – Product quality (2001)
8. Kokash, N., D'Andrea, V.: Evaluating Quality of Web Services: A Risk-Driven Approach. In: Abramowicz, W. (ed.) BIS 2007. LNCS, vol. 4439, pp. 180–194. Springer, Heidelberg (2007)
9. Laranjeiro, N., Vieira, M.: Towards fault tolerance in web services compositions. In: Guelfi, N., Muccini, H., Pelliccione, P., Romanovsky, A. (eds.) EFTS, p. 2. ACM (2007)
10. Lee, K., Jeon, J., Lee, W., Jeong, S.H., Park, S.W.: QoS for web services: Requirements and possible approaches. Tech. rep., W3C, Web Services Architecture Working Group (November 2003), `http://www.w3c.or.kr/kr-office/TR/2003/ws-qos/`

11. Ludwig, H., Keller, A., Dan, A., King, R.P., Franck, R.: Web service level agreement language specification, 1.0 (2008)
12. May, N.R.: A redundancy protocol for service-oriented architectures (2008)
13. Tian, M., Gramm, A., Ritter, H., Schiller, J.H., Voigt, T.: Qos-aware cross-layer communication for mobile web services with the ws-qos framework. In: Dadam, P., Reichert, M. (eds.) GI Jahrestagung (2). LNI, vol. 51, p. 286. GI (2004)
14. Ye, X., Mounla, R.: A hybrid approach to qos-aware service composition. In: Proceedings of the 2008 IEEE International Conference on Web Services, pp. 62–69. IEEE Computer Society, Washington, DC (2008)